Data Collection

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import math
```

In [4]:
```
td= pd.read_csv("Titanic.csv")
td.head(10)
```

Out[4]:

	PassengerID	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	

	PassengerID	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Em
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	

In [5]:
```
# No of passsengers on board Titanic

print("# of passengers in original data:" +str(len(td.index)))
```

```
# of passengers in original data:891
```

Analyzing Data

In [6]:
```
survived_count = td.groupby('Survived')['Survived'].count()
survived_count
```

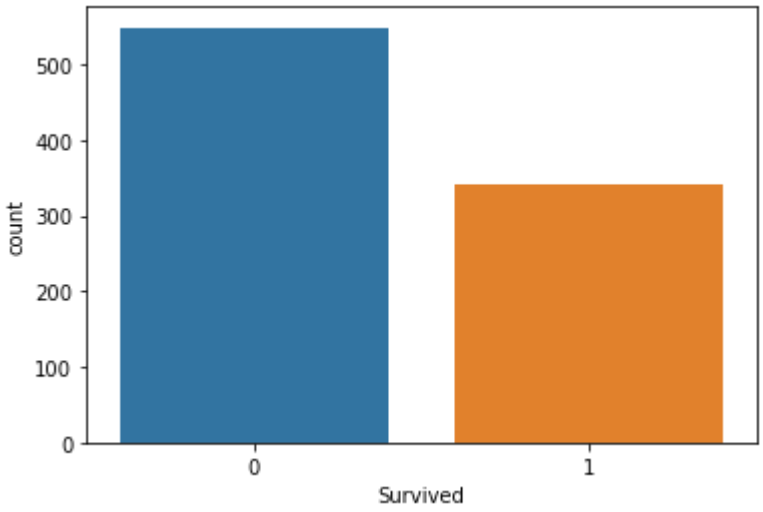
Out[6]:
```
Survived
0    549
1    342
Name: Survived, dtype: int64
```

In [7]:
```
td["Survived"].value_counts(normalize=True)
```

Out[7]:
```
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

In [8]:
```
sns.countplot(x="Survived", data=td)
```

Out[8]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [9]:
```
td.groupby('Pclass')['Pclass'].count()
```

Out[9]:
```
Pclass
1    216
2    184
```

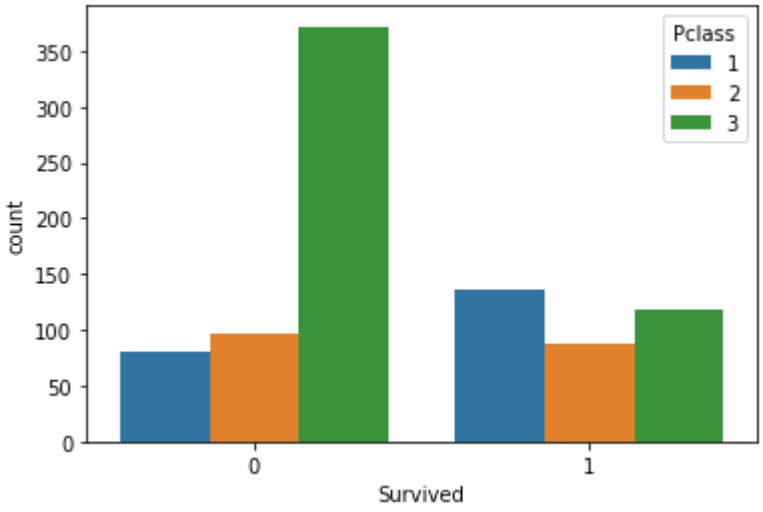
```
3    491
Name: Pclass, dtype: int64
```

In [10]:
```
td["Survived"].groupby(td["Pclass"]).mean()
```

Out[10]:
```
Pclass
1    0.629630
2    0.472826
3    0.242363
Name: Survived, dtype: float64
```

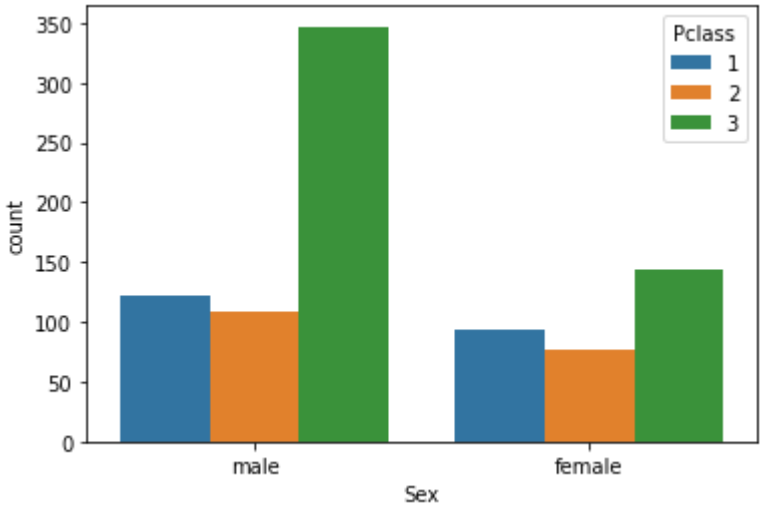
In [11]:
```
sns.countplot(x="Survived", hue="Pclass", data=td)
```

Out[11]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```



In [12]:
```
sns.countplot(x="Sex", hue="Pclass", data=td)
```

Out[12]:
```
<AxesSubplot:xlabel='Sex', ylabel='count'>
```



In [80]:
```
td.groupby('Sex')['Sex'].count()
```

Out[80]:
```
Sex
female    314
male      577
Name: Sex, dtype: int64
```

In [81]:
```
td.groupby('Sex')['Survived'].sum()
```

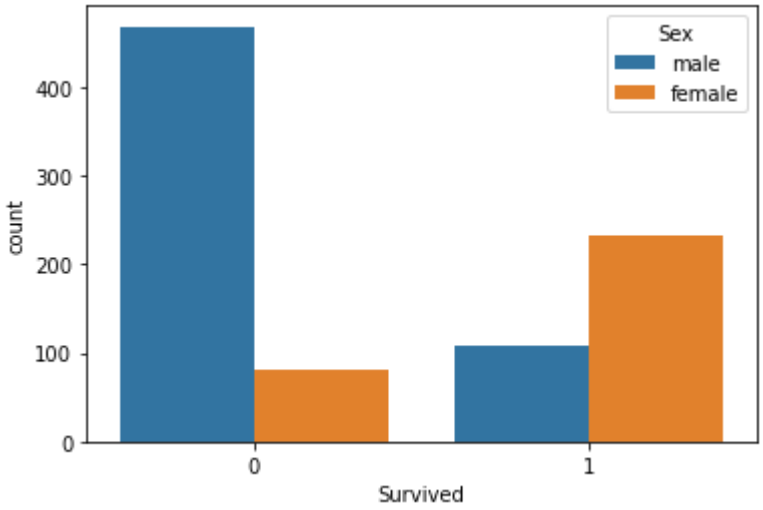
Out[81]:
```
Sex
female    233
male      109
Name: Survived, dtype: int64
```

In [82]:
```
td["Survived"].groupby(td["Sex"]).mean()
```

Out[82]:
```
Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

In [83]:
```
sns.countplot(x="Survived", hue="Sex", data=td)
```

Out[83]:
```
<AxesSubplot:xlabel='Survived', ylabel='count'>
```

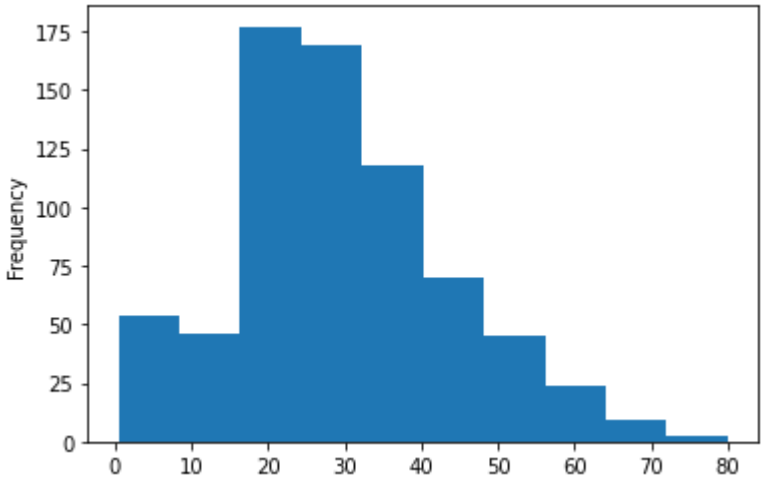


In [84]:
```
td.groupby('Age')['Age'].count()
```

Out[84]:
```
Age
0.42     1
0.67     1
0.75     2
0.83     2
0.92     1
        ..
70.00    2
70.50    1
71.00    2
74.00    1
80.00    1
Name: Age, Length: 88, dtype: int64
```

In [85]:
```
td["Age"].plot.hist()
```

Out[85]:
```
<AxesSubplot:ylabel='Frequency'>
```

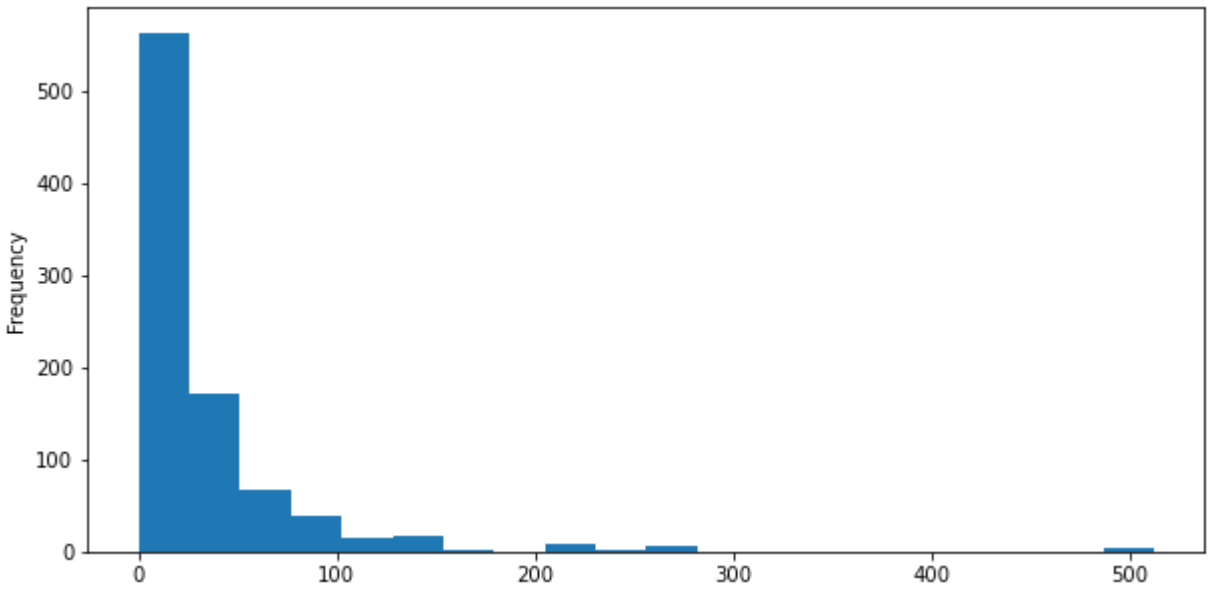


In [86]:
```
td.groupby('Fare')['Fare'].count()
```

Out[86]:
```
Fare
0.0000      15
4.0125       1
5.0000       1
6.2375       1
6.4375       1
            ..
227.5250     4
247.5208     2
262.3750     2
263.0000     4
512.3292     3
Name: Fare, Length: 248, dtype: int64
```

In [87]:
```
td["Fare"].plot.hist(bins=20, figsize=(10,5))
```

Out[87]:
```
<AxesSubplot:ylabel='Frequency'>
```



In [88]:
```
td.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column       Non-Null Count  Dtype
```

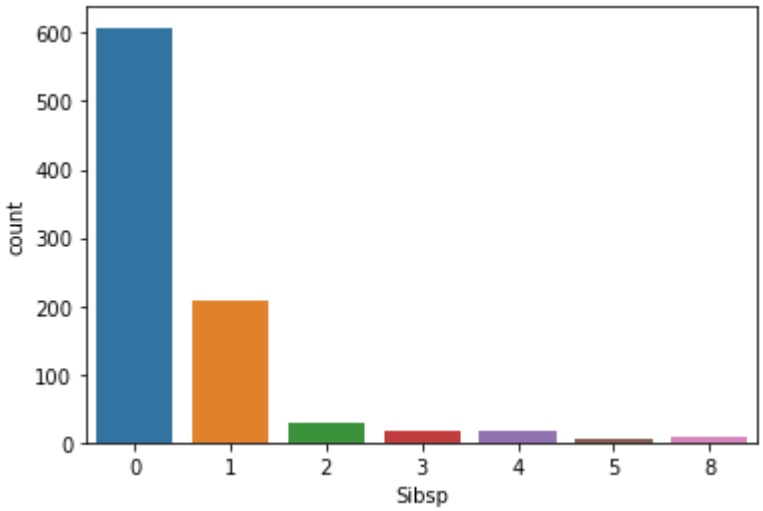
```
---  ------       --------------  -----
 0   PassengerID  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   Sibsp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
 12  Unnamed: 12  0 non-null      float64
dtypes: float64(3), int64(5), object(5)
memory usage: 90.6+ KB
```

In [89]:
```
td.groupby('Sibsp')['Sibsp'].count()
```

Out[89]:
```
Sibsp
0    608
1    209
2     28
3     16
4     18
5      5
8      7
Name: Sibsp, dtype: int64
```

In [90]:
```
sns.countplot(x="Sibsp", data=td)
```

Out[90]:
```
<AxesSubplot:xlabel='Sibsp', ylabel='count'>
```

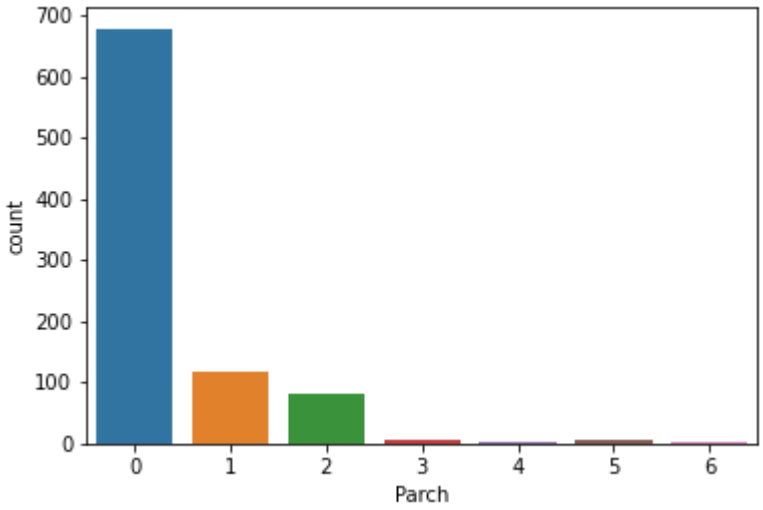


In [91]:
```
td.groupby('Parch')['Parch'].count()
```

Out[91]:
```
Parch
0    678
1    118
2     80
3      5
4      4
5      5
6      1
Name: Parch, dtype: int64
```

In [92]:
```
sns.countplot(x="Parch", data=td)
```

Out[92]: <AxesSubplot:xlabel='Parch', ylabel='count'>



Data Wrangling

In [93]:
```
td.isnull()
```

Out[93]:

	PassengerID	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Embarked
0	False	False	False	False	False	False	False	False	False	False	True	False
1	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	True	False
...	...	...	...	...	...	...	...	...	...	...	...	...
886	False	False	False	False	False	False	False	False	False	False	True	False
887	False	False	False	False	False	False	False	False	False	False	False	False
888	False	False	False	False	False	True	False	False	False	False	True	False
889	False	False	False	False	False	False	False	False	False	False	False	False
890	False	False	False	False	False	False	False	False	False	False	True	False

891 rows × 13 columns

In [94]:
```
#Processing missing data and duplicates

print('\nNull Values in td \n{}'.format(td.isnull().sum()))

print('\nDuplicated values in td {}'.format(td.duplicated().sum()))
```

```
Null Values in td
PassengerID      0
Survived         0
```

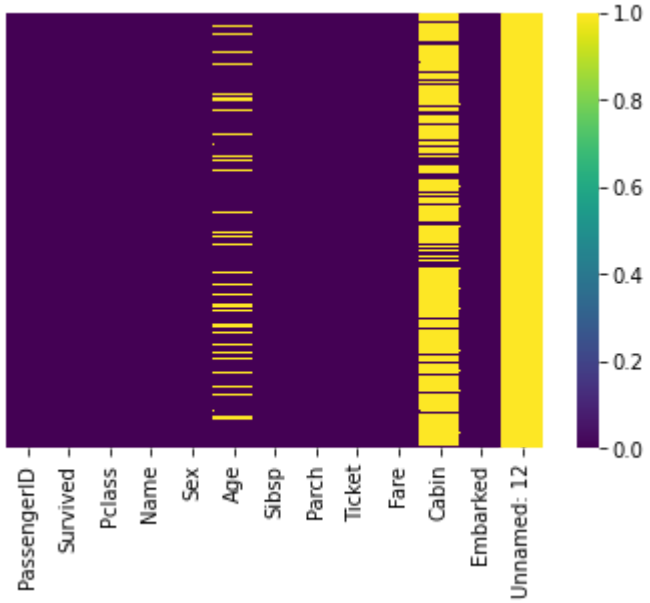
```
Pclass           0
Name             0
Sex              0
Age            177
Sibsp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
Unnamed: 12    891
dtype: int64

Duplicated values in td 0
```

In [95]:
```
# Filling Embarked

print('Embarkation per ports \n{}'.format(td['Embarked'].value_counts()))

# since the most common port is Southampton the chances are that the missing one is fro

td['Embarked'].fillna(value='S', inplace=True)

print('Embarkation per ports after filling \n{}'.format(td['Embarked'].value_counts()))
```

```
Embarkation per ports
S    644
C    168
Q     77
Name: Embarked, dtype: int64
Embarkation per ports after filling
S    646
C    168
Q     77
Name: Embarked, dtype: int64
```

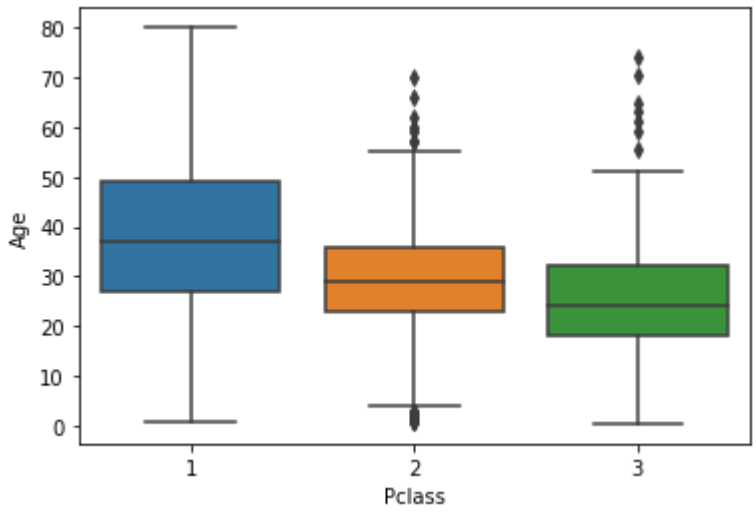
In [96]:
```
sns.heatmap(td.isnull(), yticklabels=False, cmap="viridis")
```

Out[96]: <AxesSubplot:>



In [97]:
```
sns.boxplot(x="Pclass", y="Age", data=td)
```

Out[97]: <AxesSubplot:xlabel='Pclass', ylabel='Age'>



In [98]:
```
td.head(5)
```

Out[98]:

	PassengerID	Survived	Pclass	Name	Sex	Age	Sibsp	Parch	Ticket	Fare	Cabin	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	

In [99]:
```
#Drop features which are not important the model,name,cabin,ticket,unnamed: 12

td = td.drop(["Name", "Cabin", "Ticket", "Unnamed: 12"], axis=1)
```

In [100…
```
td.head(5)
```

Out[100…

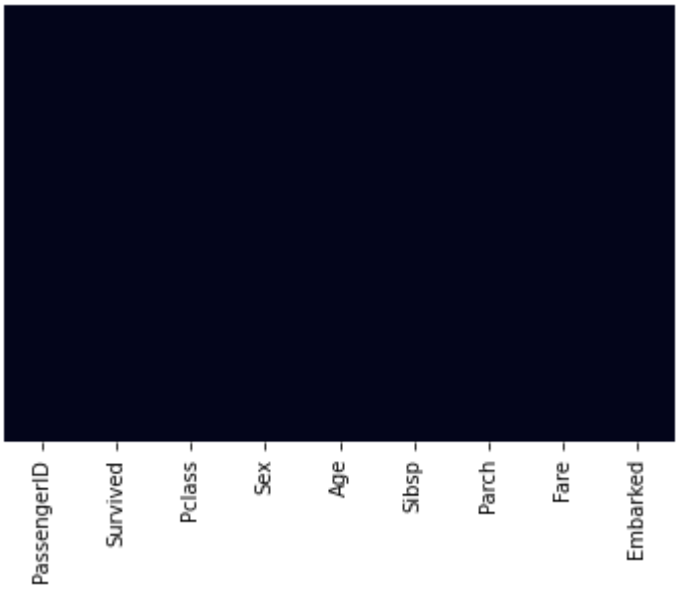
	PassengerID	Survived	Pclass	Sex	Age	Sibsp	Parch	Fare	Embarked
0	1	0	3	male	22.0	1	0	7.2500	S

	PassengerID	Survived	Pclass	Sex	Age	Sibsp	Parch	Fare	Embarked
1	2	1	1	female	38.0	1	0	71.2833	C
2	3	1	3	female	26.0	0	0	7.9250	S
3	4	1	1	female	35.0	1	0	53.1000	S
4	5	0	3	male	35.0	0	0	8.0500	S

In [101…
```
td.dropna(inplace=True)
```

In [102…
```
sns.heatmap(td.isnull(), yticklabels=False, cbar=False)
```

Out[102… <AxesSubplot:>



In [103…
```
td.isnull().sum()
```

Out[103…
```
PassengerID    0
Survived       0
Pclass         0
Sex            0
Age            0
Sibsp          0
Parch          0
Fare           0
Embarked       0
dtype: int64
```

In [104…
```
sex=pd.get_dummies(td["Sex"],drop_first=True)
sex.head()
```

Out[104…

	male
0	1
1	0
2	0
3	0

	male
4	1

In [105…
```
embark=pd.get_dummies(td["Embarked"],drop_first=True)
embark.head()
```

Out[105…

	Q	S
0	0	1
1	0	0
2	0	1
3	0	1
4	0	1

In [106…
```
Pcl=pd.get_dummies(td["Pclass"],drop_first=True)
Pcl.head()
```

Out[106…

	2	3
0	0	1
1	0	0
2	0	1
3	0	0
4	0	1

In [107…
```
td=pd.concat([td, sex,embark,Pcl], axis=1)
```

In [108…
```
td.head(5)
```

Out[108…

	PassengerID	Survived	Pclass	Sex	Age	Sibsp	Parch	Fare	Embarked	male	Q	S	2	3
0	1	0	3	male	22.0	1	0	7.2500	S	1	0	1	0	1
1	2	1	1	female	38.0	1	0	71.2833	C	0	0	0	0	0
2	3	1	3	female	26.0	0	0	7.9250	S	0	0	1	0	1
3	4	1	1	female	35.0	1	0	53.1000	S	0	0	1	0	0
4	5	0	3	male	35.0	0	0	8.0500	S	1	0	1	0	1

In [109…
```
#Drop features which are not important the model: PassengerID,Pclass,Sex,Embarked


td = td.drop(["PassengerID", "Pclass", "Sex", "Embarked"], axis=1)
```

In [110…
```
td.head(5)
```

Out[110…

	Survived	Age	Sibsp	Parch	Fare	male	Q	S	2	3

	Survived	Age	Sibsp	Parch	Fare	male	Q	S	2	3
0	0	22.0	1	0	7.2500	1	0	1	0	1
1	1	38.0	1	0	71.2833	0	0	0	0	0
2	1	26.0	0	0	7.9250	0	0	1	0	1
3	1	35.0	1	0	53.1000	0	0	1	0	0
4	0	35.0	0	0	8.0500	1	0	1	0	1

Train & Test Data using Logistic Regression

In [111…
```
# Import libraries required
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [112…
```
# 1) Separate the data into independent and dependent variables

x= td.drop("Survived", axis=1)
y= td["Survived"]
```

In [113…
```
# 2) Train test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state
```

In [114…
```
# Train the model

log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

```
C:\Users\Rose Yong\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:762: Co
nvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[114…
```
LogisticRegression()
```

In [115…
```
# Predict the model

predictions = log_reg.predict(x_test)
```

In [116…
```
from sklearn.metrics import classification_report
```

In [117…
```
# Evaluate the accuracy of the model

classification_report(y_test,predictions)
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.80      0.87      0.83        86
           1       0.78      0.67      0.72        57
```

```
              accuracy                           0.79       143
             macro avg       0.79      0.77      0.78       143
          weighted avg       0.79      0.79      0.79       143
```

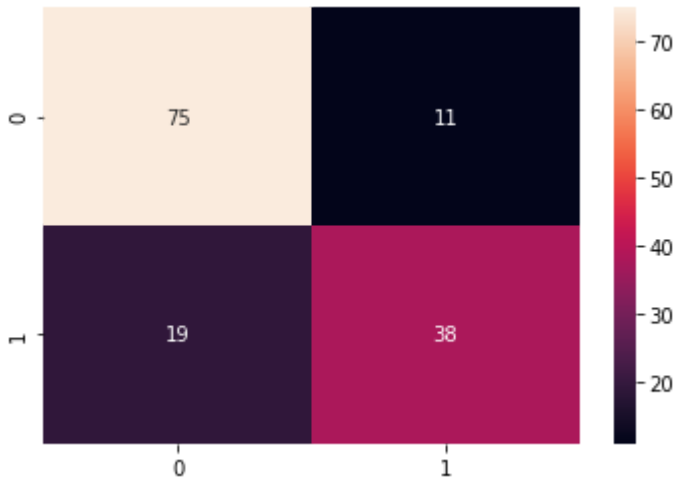
In [120…
```
from sklearn.metrics import confusion_matrix
```

In [121…
```
# Confusion Matrix

conf_mat = confusion_matrix(y_test, predictions)
print("Confusion Matrix = \n", str(conf_mat))
```

```
Confusion Matrix =
 [[75 11]
 [19 38]]
```

In [122…
```
import seaborn as sns
sns.heatmap(conf_mat, annot=True)
```

Out[122… <AxesSubplot:>



In [123…
```
# import the libraries
from sklearn.metrics import accuracy_score
```

In [124…
```
# Accuracy
test_acc = accuracy_score(y_test, predictions)

print("Testing Accuracy = ", str(test_acc))
print("Testing Accuracy = {}".format(test_acc))
```

```
Testing Accuracy =  0.7902097902097902
Testing Accuracy = 0.7902097902097902
```

Train & Test Data using Random Forest

In [120…
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix , classification_report
```

In [121…
```
# Split the Data
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_stat
```

In [123…
```
# Initialize and train the machine learning model

rf_classifier = RandomForestClassifier()
rf_classifier.fit(x_train, y_train)
```

Out[123…
```
RandomForestClassifier()
```

In [124…
```
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
from sklearn.model_selection import GridSearchCV

param_grid = {'criterion': ['gini', 'entropy'],
              'max_depth': range(1,10),
              'min_samples_split': range(1, 10),
              'min_samples_leaf': range(1, 5)}

gs_rf = GridSearchCV(clf,
                     param_grid,
                     cv=5,
                     scoring='f1_macro',
                     n_jobs=-1)
gs_rf.fit(x_train, y_train)
```

Out[124…
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 10),
                         'min_samples_leaf': range(1, 5),
                         'min_samples_split': range(1, 10)},
             scoring='f1_macro')
```

In [125…
```
# These are the best parameters for the decision tree classifier
gs_rf.best_estimator_
```

Out[125…
```
DecisionTreeClassifier(max_depth=9, min_samples_leaf=4, min_samples_split=9,
                       random_state=42)
```

In [126…
```
# Training the final model using the best parameters from above fine-tuning
classifier = gs_rf.best_estimator_
classifier.fit(x_train, y_train)
```

Out[126…
```
DecisionTreeClassifier(max_depth=9, min_samples_leaf=4, min_samples_split=9,
                       random_state=42)
```

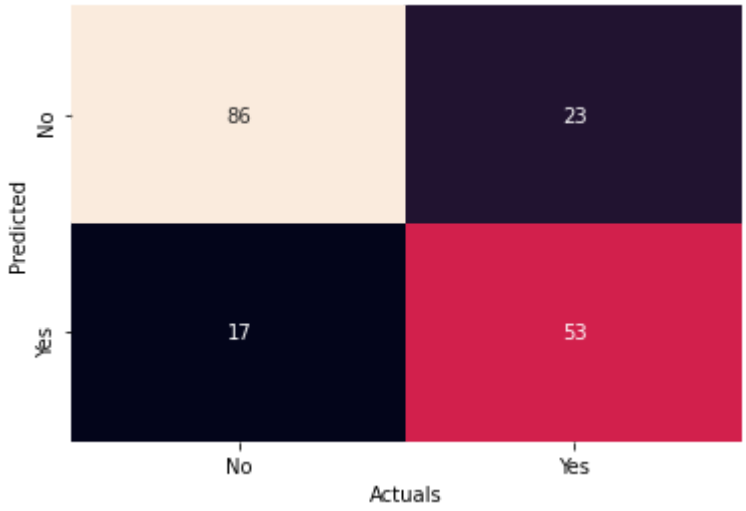
In [129…
```
# Predict the model

y_pred = rf_classifier.predict(x_test)
```

In [130…
```
# Classification Report

print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       103
           1       0.76      0.70      0.73        76

    accuracy                           0.78       179
   macro avg       0.77      0.77      0.77       179
weighted avg       0.78      0.78      0.78       179
```

In [131…
```
## Confusion Matrix

conf_mat = confusion_matrix(y_test, y_pred)
sns.heatmap(conf_mat.T, annot=True, fmt='d', cbar=False,
          xticklabels=['No','Yes'],
          yticklabels=['No','Yes'] )
plt.xlabel('Actuals')
plt.ylabel('Predicted')
```

Out[131…
```
Text(33.0, 0.5, 'Predicted')
```



In [132…
```
# import the libraries
from sklearn.metrics import accuracy_score
```

In [133…
```
# Accuracy
test_acc = accuracy_score(y_test, y_pred)

print("Testing Accuracy = ", str(test_acc))
print("Testing Accuracy = {}".format(test_acc))
```

```
Testing Accuracy =  0.776536312849162
Testing Accuracy = 0.776536312849162
```

Train & Test Data using Decision Tree

In [66]:
```
#  Splitting our data into training and testing sets

from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x ,y,
                                                    test_size = 0.2,
                                                    shuffle=True,
                                                    stratify=y,
                                                    random_state=42)
```

In [67]:
```
# Import the Decision Tree algorithm
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=42)
```

In [70]:
```
# Hyperparameter tuning using K-fold cross validation
# ... via Grid Search method
```

```
from sklearn.model_selection import GridSearchCV

param_grid = {'criterion': ['gini', 'entropy'],
              'max_depth': range(1,10),
              'min_samples_split': range(1, 10),
              'min_samples_leaf': range(1, 5)}

gs_clf = GridSearchCV(clf,
                      param_grid,
                      cv=5,
                      scoring='f1_macro',
                      n_jobs=-1)
gs_clf.fit(x_train, y_train)
```

Out[70]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=42), n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 10),
                         'min_samples_leaf': range(1, 5),
                         'min_samples_split': range(1, 10)},
             scoring='f1_macro')
```

In [71]:
```
# These are the best parameters for the decision tree classifier

gs_clf.best_estimator_
```

Out[71]:
```
DecisionTreeClassifier(max_depth=5, min_samples_split=5, random_state=42)
```

In [73]:
```
# Training the final model using the best parameters from above fine-tuning

classifier = gs_clf.best_estimator_
classifier.fit(x_train, y_train)
```

Out[73]:
```
DecisionTreeClassifier(max_depth=5, min_samples_split=5, random_state=42)
```

In [75]:
```
# Kept aside some data to test - X_test

y_pred = classifier.predict(x_test)
```

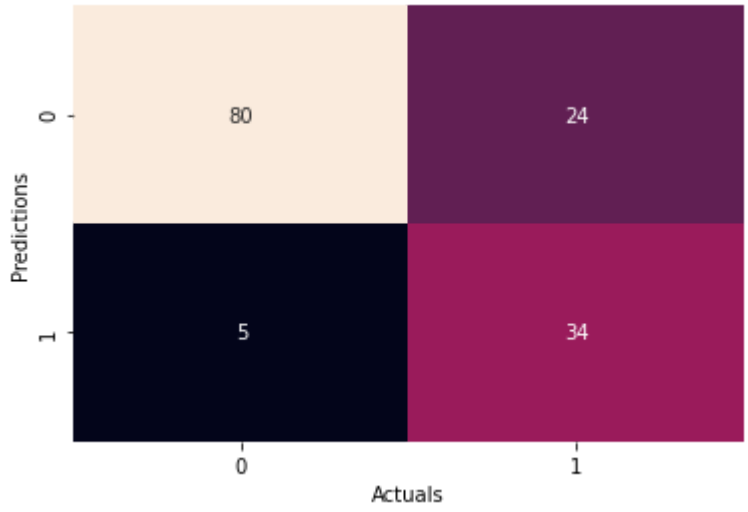
In [76]:
```
from sklearn.metrics import classification_report, confusion_matrix

# Classification Report
print("Classification report:")
print()
print(classification_report(y_test, y_pred, target_names=['0','1']))
print()
```

```
Classification report:

              precision    recall  f1-score   support

           0       0.77      0.94      0.85        85
           1       0.87      0.59      0.70        58

    accuracy                           0.80       143
   macro avg       0.82      0.76      0.77       143
weighted avg       0.81      0.80      0.79       143
```

In [77]:
```
# Confusion Matrix
```

```
conf_mat = confusion_matrix(y_test, y_pred)

import seaborn as sns
sns.heatmap(conf_mat.T, annot=True, cbar=False)
plt.xlabel("Actuals")
plt.ylabel("Predictions")
```

Out[77]: Text(33.0, 0.5, 'Predictions')



In [78]:
```
# import the libraries
from sklearn.metrics import accuracy_score
```

In [79]:
```
# Accuracy
test_acc = accuracy_score(y_test, y_pred)

print("Testing Accuracy = ", str(test_acc))
print("Testing Accuracy = {}".format(test_acc))
```

```
Testing Accuracy =  0.7972027972027972
Testing Accuracy = 0.7972027972027972
```

In [ ]: